Fall 2011

Interface Design for Graphics Editor on Multi- Touch Point Systems

Srujitha Mullapudi
San Jose State University

Recommended Citation

Mullapudi, Srujitha, "Interface Design for Graphics Editor on Multi- Touch Point Systems" (2011). *Master's Projects*. 199.
DOI: https://doi.org/10.31979/etd.28gt-9j9j
https://scholarworks.sjsu.edu/etd_projects/199

Interface Design for Graphics Editor on Multi-Touch Point Systems

A Writing Project

Presented to

The Faculty of the department of Computer

Science San Jose State University

In Partial Fulfillment

of the Requirements for the

Degree Master of Computer Science

By
Srujitha Mullapudi
Oct 2011

SAN JOSE STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

Interface Design for Graphics Editor on Multi-Touch Point Systems

By

Srujitha Mullapudi

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

___

Dr. Soon Tee Teoh, Department of Computer Science 11/30 /2011

___

Dr. Mark Stamp, Department of Computer Science 11/30 /2011

___

Ms. Pooja Palan, ModelN corporaton 11/ 30/2011

ACKNOWLEDGEMENTS

I am very thankful to my project advisor, Dr. Soon Tee Teoh, for his support and suggestions throughout my master's project and also I would like to thank my committee members, Dr. Mark Stamp, and Ms. Pooja Palan for their time and effort.

I would like to thank my parents and my brother, for their encouragement and constant support. I would also like to thank my friends to give me a valuable feedback on my project.

ABSTRACT

Interface Design for Graphics Editor on Multi-Touch Point Systems

The main objective of the project is to use the touch sensitive device capability of the iPad to give a better user experience and functionalities for the artists. In this project, I have explored different interactions that can give a better experience for artist when compared to traditional computer-mouse interaction. In the traditional Computer-mouse interaction it is very difficult for the user to draw smooth curves, without having to use built-in functions for drawing curves. In this application the user can use the touch panel to do free drawing, the user can basically feel the device as a object for drawing like in real life. Drawing on the touch device eliminates a use of another hardware to interact with the Interface of the system. In this application the user can use multi-touch points to zoom and pan, the user can also choose colors from a color wheel, the changable pen and eraser size. The pen size is varied based on the touch of the user touch radius. The Important functionality of the project is, the application lets the user to obtain subtractive color blending. User can also add a image to the background from the photo library and add more colors to the image. User can also zoom and draw, which adds the advantage of being able to modify the minute details of the drawing. Different actions can be performed on the image drawn, like copy, save to gallery, email and different services are also built into the app to allow the users to share the image to online services like Facebook, Twitter and Tumblr.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Project Overview

“Computer Graphics started with the display of data on hardcopy plotters and cathode ray (CRT) screens soon after introduction of computers themselves. It has grown to include the creation, storage, and manipulation of models and images of objects. The user controls the contents, structure and appearance of objects and of their displayed images by using input devices, such as keyboard, mouse, or touch-sensitive panel on the screen” [7]. There is a close relationship between graphics display and input devices. It is very inconvenient for a user to have a free style drawing using the mouse or keyboard.

It would be easier if a user can feel the display as an object to Interact directly on it, which is possible through touch-sensitive panels. There are a lot of touch-sensitive devices currently available in the market, and software’s that allow users to interact without the need of any other input device, as the display itself acts as input device. Sketching on a digital device has become very popular recently, due to large and easy availability of the touch-sensitive devices in the market.

In this project the application developed on a multi-touch sensitive iPad allows the user to draw with touch, different attributes relating to a touch are being taken advantage of, like radius of the touch, continuous touch points to draw a curve based on the user movement of the touch. The application uses OpenGL ES 2.0 on iPad for drawing. The advantage of OpenGL ES for drawing is the texture can be created using an image file and rendering it in a framerbuffer object before displaying it on the screen. This allows the application to create a brush type available from an image file, rendering it with the color selected.

In this application the user can use multi-touch points to zoom and pan, the user can also choose colors from a color wheel, the changable pen and eraser size. The pen size is varied based on the touch of the user touch radius. The Important functionality of the project is, the application lets the user to obtain subtractive color blending. User can also add a image to the background from the photo library and add more colors to the image. User can also zoom and draw, which adds the advantage of being able to modify the minute details of the drawing. Different actions can be performed on the image drawn, like copy, save to gallery, email and different services are also built into the app to allow the users to share the image to online services like Facebook, Twitter and Tumblr. In this application the user interface is designed in a way that empower users to be not only more productive, but more innovative.

2. Architecture

2.1 iOS Architecture

"In order to understand the application development, it is important to understand the iOS architecture. iOS is an operating system for handheld devices manufactured by Apple. It runs on iPhone, iPod touch, and iPad devices. Like any other operating system it manages the device hardware and provides required technologies to implement applications. The operating system also has built in various system applications for checking emails, web browsing that provide standard system services to the user" [4].

Apple also provides the developers with the required SDK for application development, which also includes tools and interfaces that makes it convenient for developer to build rich applications. It also includes simulator that can be used for testing the application prior to testing on the device. The testing on the device requires apple developer membership which can be obtained with a certain amount of yearly membership fee. There are two different kinds of applications that can be built on the iOS based device, Native applications are built using the iOS system frameworks and Objective-C language and run directly on iOS, the other type is web based applications, native applications are installed physically on a device and are therefore always available to the user, even when the device is in Airplane mode. This project is a Native application which takes the advantage of OpenGL ES framework built into the iOS.

“At the highest level, iOS acts as an intermediary between the underlying hardware and the applications that appear on the screen, as shown in Figure 2.1.1. The applications rarely talk to the underlying hardware directly. Instead, applications communicate with the hardware through a set of well-defined system interfaces that protect the application from hardware changes. This abstraction makes it easy to write applications that work consistently on devices with different hardware capabilities.” [4]

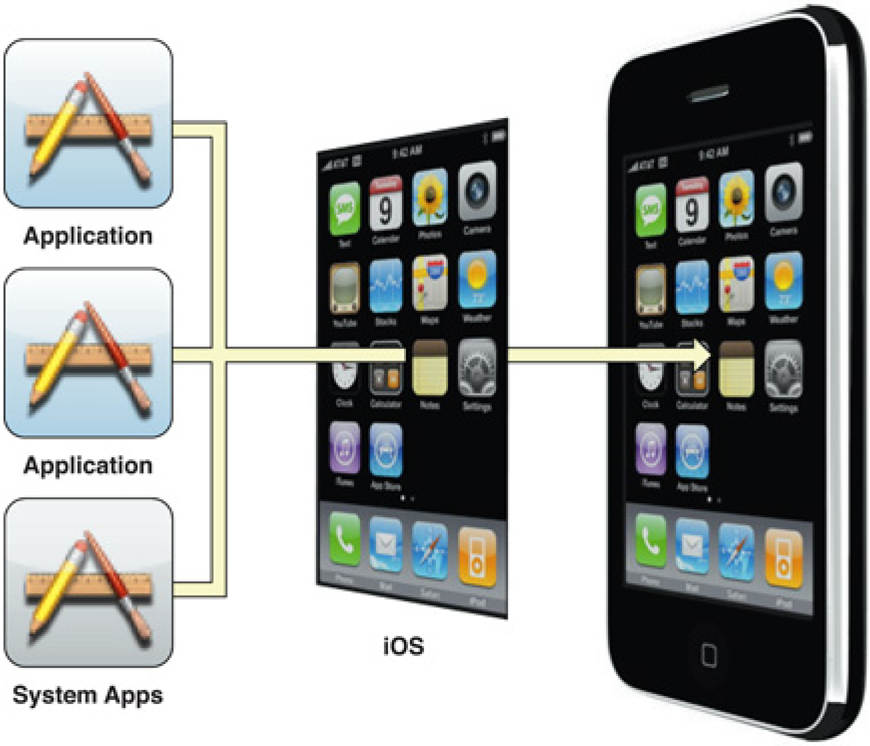


Figure 2.1.1 Applications layered on top of iOS

“The implementation of iOS technologies can be viewed as a set of layers, which are shown in Figure 2.1.2. At the lower layers of the system are the fundamental services and technologies on which all applications rely. Higher-level layers contain more sophisticated services and technologies.

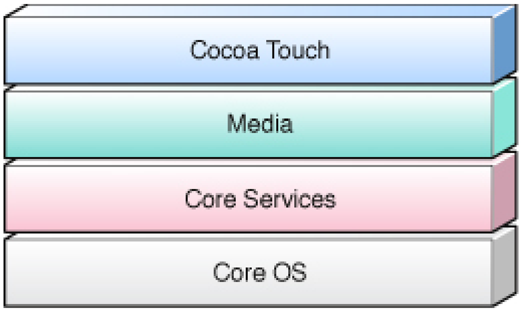


Figure 2.1.2 Layers of iOS

sThe higher-level frameworks are there to provide object-oriented abstractions for lower-level constructs . These abstractions generally make it much easier to write code because they reduce the amount of code for the developers to write and encapsulate potentially complex features, such as sockets and threads . The lower-level frameworks are still available for developers who prefer to use them or who want to use aspects of those frameworks that are not exposed by the higher layers" [4].

2.2 OpenGL ES

“The Open Graphics Library (OpenGL) is used for visualizing 2D and 3D data. It is a multipurpose open-standard graphics library that supports applications for 2D and 3D digital content creation, mechanical and architectural design, virtual prototyping, flight simulation, video games, and more. It allows developers to configure a 3D and 2D graphics pipeline and submit data to it. Vertices are transformed and lit, assembled into primitives, and rasterized to create a 2D image” [4]. It is designed to communicate with the underlying graphics hardware by translating the function calls into graphics commands. Due to the dedicated hardware for processing graphics commands, OpenGL ES drawing is very fast.

“OpenGL for Embedded Systems (OpenGL ES) is a simplified version of OpenGL that eliminates redundant functionality to provide a library that is both easier to learn and easier to implement in mobile graphics hardware” [4].

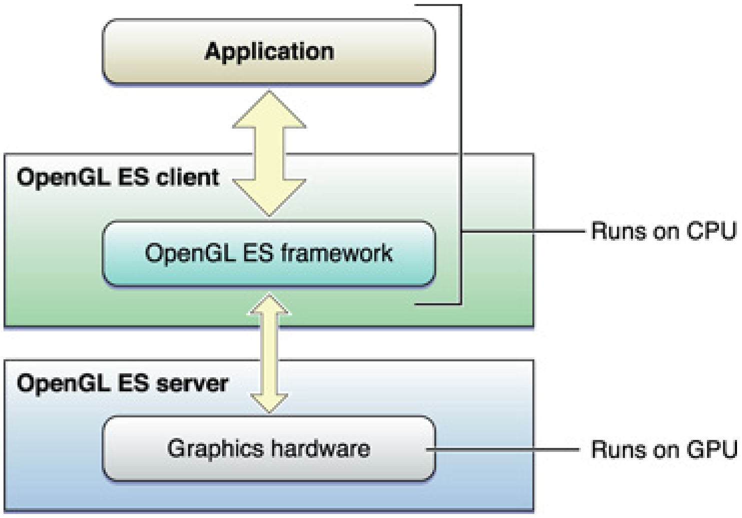


Figure 2.2.1 – OpenGL ES framework.

“ The most important OpenGL ES object types include:

- A **texture** is an image that can be sampled by the graphics pipeline. It is used to map an image onto primitives, other data such as pre-calculated map or normal map can also be mapped.
- A **buffer** object is a block of memory owned by OpenGL ES used to store data for the application. Buffers are used to precisely control the process of copying data between the application and OpenGL ES. For example, if you provide a vertex array to OpenGL ES, it must copy the data every time you submit a drawing call. In contrast, if the application stores its data in a *vertex buffer object*, the data is copied only when the application sends commands to modify the contents of the vertex buffer object. Using buffers to manage the vertex data can significantly boost the performance of the application.
- A **vertex array** object holds a configuration for the vertex attributes that are to be read by the graphics pipeline. Many applications require different pipeline configurations for each entity it intends to render. By storing a configuration in a vertex array, you avoid the cost of reconfiguring the pipeline and may allow the implementation to optimize its handling of that particular vertex configuration.
- **Shader programs**, also known as shaders, are also objects. An OpenGL ES 2.0 application creates vertex and fragment shaders to specify the calculations that are to be performed on each vertex or fragment, respectively.
- A **renderbuffer** is a simple 2D graphics image in a specified format. This format usually is defined as color, depth or stencil data. Renderbuffers are not usually used in isolation, but are instead used as attachments to a framebuffer.

- **Framebuffer** objects are the ultimate destination of the graphics pipeline. A framebuffer object is really just a container that attaches textures and renderbuffers to itself to create a complete configuration needed by the renderer.” [4]

2.2.1 Framebuffer objects are the only rendering targets in OpenGL ES

“Framebuffer objects are the destination for rendering commands. OpenGL ES 2.0 provides framebuffer objects as part of the core specification. Framebuffer objects provide storage for color, depth and/or stencil data by attaching images to the framebuffer, as shown in Figure 2.2.3. The most common image attachment is a renderbuffer object. However, an OpenGL ES texture can be attached to the color attachment point of a framebuffer instead, allowing image to be rendered directly into a texture. Later, the texture can act as an input to future rendering commands.” [4]

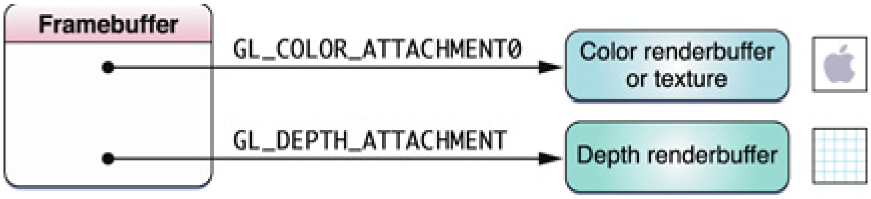


Figure 2.2.2 Framebuffer with color and depth renderbuffers

3. Implementation

Application implementation involves complex steps, In order to draw on OpenGL ES framework, framebuffer objects need to be created and updating the framebuffer objects based on the user drawing.

3.1 Creating Framebuffer Objects

The OpenGL ES specification requires that each implementation provide a mechanism that an application can use to create a framebuffer to hold rendered images. On iOS, all framebuffers are implemented using framebuffer objects, which are built-in to OpenGL ES 2.0.

"Framebuffer objects allow the application to precisely control the creation of color, depth, and stencil targets" [4]. It also allows creating multiple framebuffer objects on a single context, possibly sharing resources between the frame buffers.

The procedure to create a framebuffer as follows:

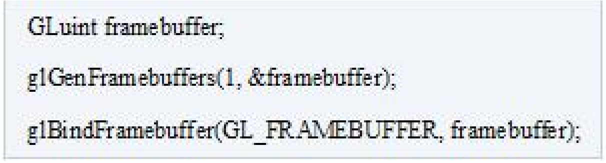
1. Create a framebuffer object.
2. Create one or more targets (renderbuffers or textures), allocate storage for them, and attach each to an attachment point on the framebuffer object.
3. Test the framebuffer for completeness.

Depending on what task the application intends to perform, the application configures different objects to attach to the framebuffer object. In most cases, the difference in configuring the framebuffer is in what object is attached to the framebuffer object's color attachment point.

3.1.1 Creating Offscreen Framebuffer Objects

A framebuffer intended for off-screen rendering allocates all of its attachments as OpenGL ES renderbuffers. The following code allocates a framebuffer object with color and depth attachments.

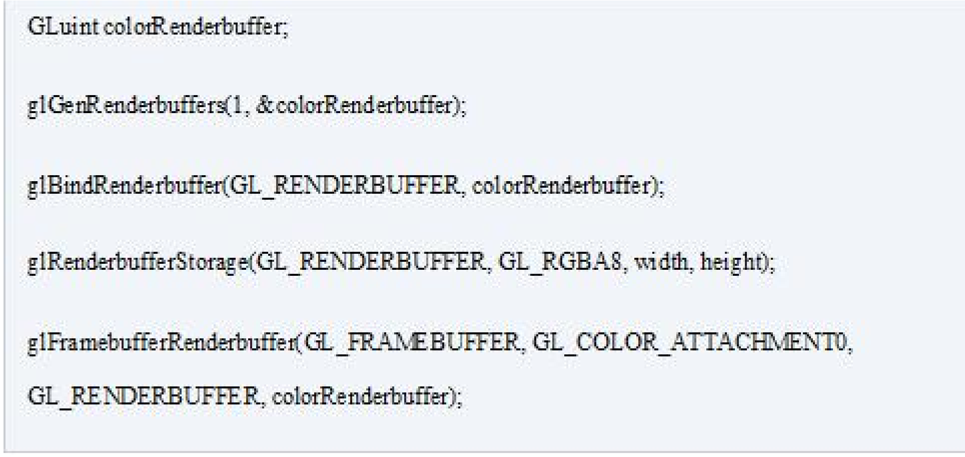
1. Create the framebuffer and bind it.



```
GLuint framebuffer;
glGenFramebuffers(1, &framebuffer);
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
```

Table 3.1.1 Create the framebuffer and bind it.

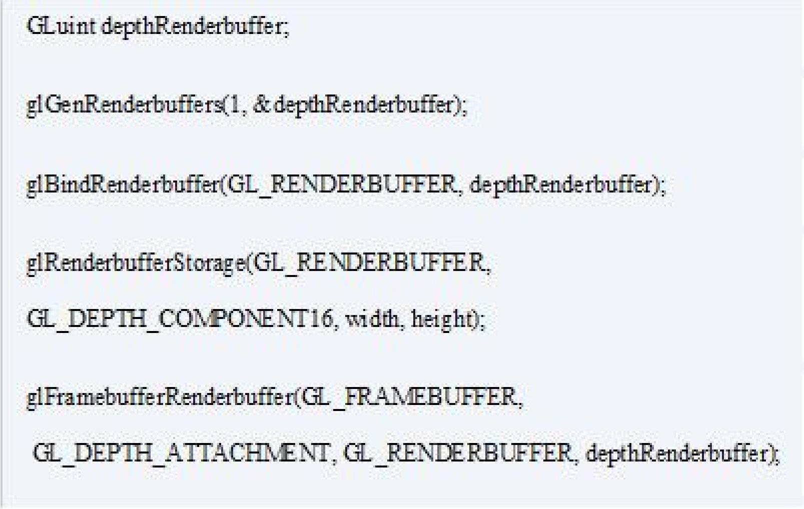
2. Create a color renderbuffer, allocate storage for it, and attach it to the framebuffer's color attachment point.



```
GLuint colorRenderbuffer;

glGenRenderbuffers(1, &colorRenderbuffer);

glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);

glRenderbufferStorage(GL_RENDERBUFFER, GL_RGBA8, width, height);

glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
GL_RENDERBUFFER, colorRenderbuffer);
```

Table 3.1.2 Create a color renderbuffer

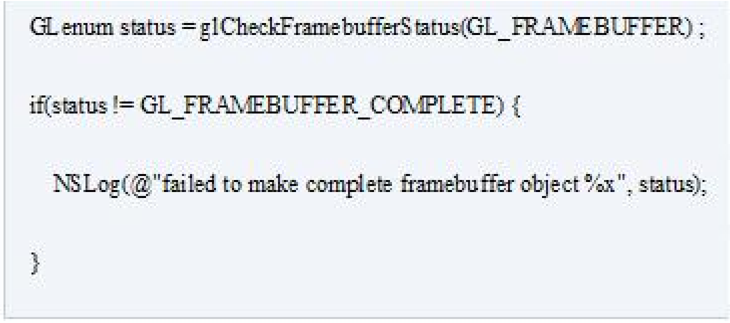
3. “Create a depth or depth/stencil renderbuffer, allocate storage for it, and attach it to the framebuffer’s depth attachment point” [4].



```
GLuint depthRenderbuffer;

glGenRenderbuffers(1, &depthRenderbuffer);

glBindRenderbuffer(GL_RENDERBUFFER, depthRenderbuffer);

glRenderbufferStorage(GL_RENDERBUFFER,
GL_DEPTH_COMPONENT16, width, height);

glFramebufferRenderbuffer(GL_FRAMEBUFFER,
 GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, depthRenderbuffer);
```

Table 3.1.3 Create depth/stencil renderbuffer

4. Testing the framebuffer object for completeness. The test will only be needed each time when the framebuffer’s configuration changes.



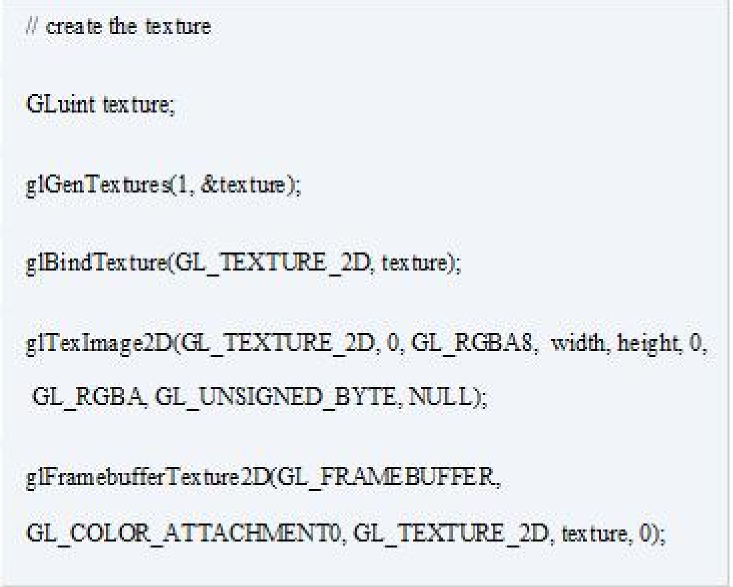
```
GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER) ;

if(status != GL_FRAMEBUFFER_COMPLETE) {

   NSLog(@"failed to make complete framebuffer object %x", status);

}
```

Table 3.1.4 Test the framebuffer

3.1.2 Using Framebuffer Objects to Render to a Texture

In this application the framebuffer is used as an input to a later rendering step, a texture is allocated and attached to the color attachment point. Every framebuffer allocates a color attachment and a depth attachment, the only primary difference is how the color attachment is allocated.

1. Create the framebuffer object.
2. Create the destination texture, and attach it to the framebuffer’s color attachment point.



```
// create the texture

GLuint texture;

glGenTextures(1, &texture);

glBindTexture(GL_TEXTURE_2D, texture);

glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8,  width, height, 0,
 GL_RGBA, GL_UNSIGNED_BYTE, NULL);

glFramebufferTexture2D(GL_FRAMEBUFFER,
GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texture, 0);
```

Table 3.1.5 Create the destination texture

3. Allocate and attach a depth buffer.
4. Test the framebuffer for completeness.

3.1.3 Rendering to a Core Animation Layer

Most applications that draw using OpenGL ES want to display the contents of the framebuffer to the user. On iOS, all images displayed on the screen are handled by Core Animation. Every view is backed by a corresponding Core Animation layer. OpenGL ES connects to Core Animation through a special Core Animation layer, a CAEAGLLayer.

A CAEAGLLayer allows the contents of an OpenGL ES renderbuffer to also act as the contents of a Core Animation layer. This allows the renderbuffer contents to be transformed and composited with other layer content, including content rendered using UIKit or Quartz. Once Core Animation composites the final image, it is displayed on the device's main screen or an attached external display.

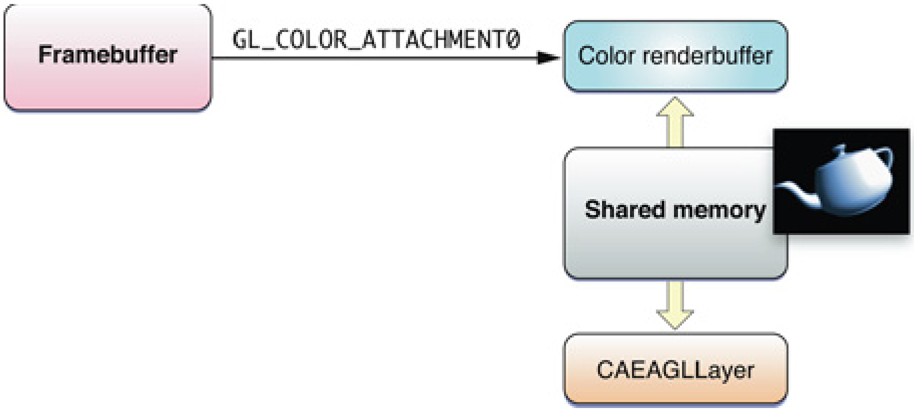


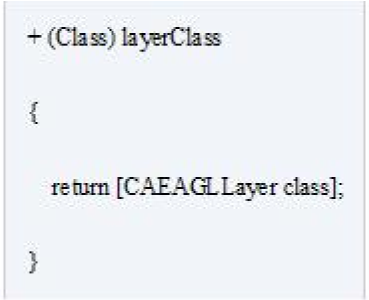
Figure 3.1.1 Core Animation shares the renderbuffer with OpenGL ES

In most cases, the application never directly allocates a CAEAGLLayer object. Instead, the application defines a subclass of UIView that allocates a CAEAGLLayer object as its backing layer. At runtime, the application instantiates the view places it into the view hierarchy. When the view is instantiated, the application initializes an OpenGL ES context and creates a framebuffer object that connects to Core Animation.

The CAEAGLLayer provides this support to OpenGL ES by implementing the EAGLDrawable protocol. An object that implements the EAGLDrawable works closely with an EAGLContext object. A drawable object provides two key pieces of functionality. First, it allocates shared storage for a renderbuffer. Second, it works closely with the context to *present* that renderbuffer's content. Presenting the contents of a renderbuffer is loosely defined by EAGL; for a CAEAGLLayer object, presenting means that the renderbuffer's contents replace any previous contents presented to Core Animation. An advantage of this model is that the contents of the Core Animation layer do not need to be rendered every frame, only when the rendered image would actually change.

It is illustrative to walk through the steps used to create an OpenGL ES-aware view. The OpenGL ES template provided by Xcode implements this code for you.

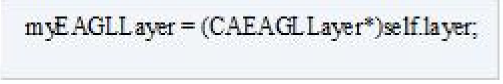
1. Subclass UIView to create an OpenGL ES view for the iOS application.
2. Override the layerClass method so that the view creates a CAEAGLLayer object as its underlying layer. To do this, the layerClass method returns theCAEAGLLayer class.



```
+ (Class) layerClass
{
   return [CAEAGLLayer class];
}
```

Table 3.1.6 Override the layerclass

3. In the view’s initialization routine, read the layer property of the view. The code uses this when it creates the framebuffer object.



```
myEAGLLayer = (CAEAGLLayer*)self.layer;
```

Table 3.1.7 Read layer properties

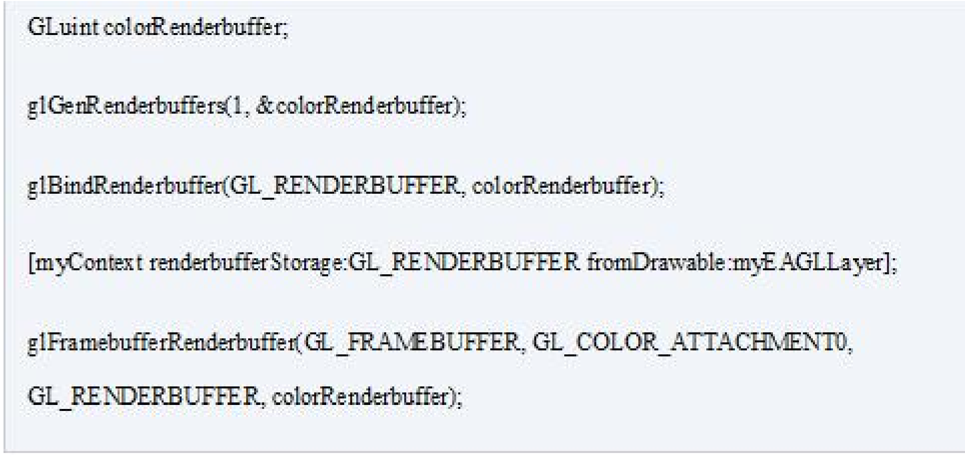
4. Configure the layer’s properties. For optimal performance, mark the layer as opaque by setting the opaque property provided by the CALayer class to YES. Optionally, configure the surface properties of the rendering surface by assigning a new dictionary of values to the drawableProperties property of theCAEAGLLayer object. EAGL allows you to specify the pixel format for the renderbuffer and whether it retains its contents after they are presented to the Core Animation.

5. Allocate a context and make it the current context.

6. Create the framebuffer object as above.

7. Create a color renderbuffer. Allocating its storage by calling the context’s renderbufferStorage:fromDrawable: method, passing the layer object as the parameter.

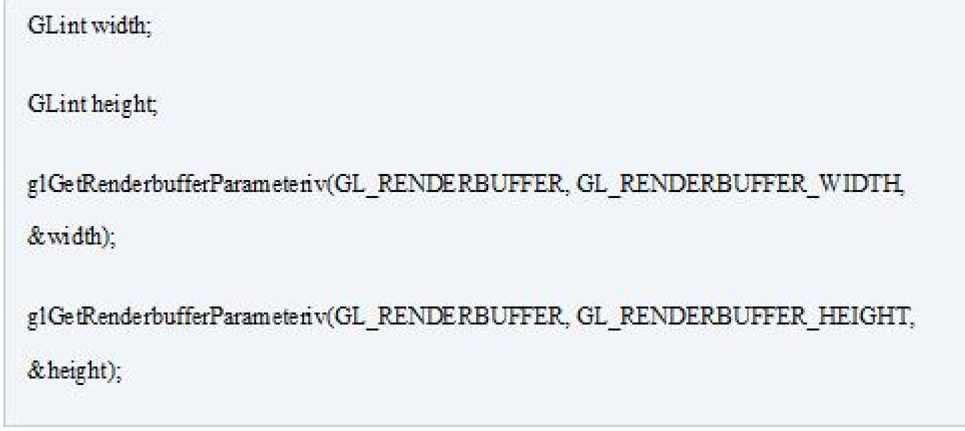
The width, height and pixel format are taken from the layer and used to allocate storage for the renderbuffer.



```
GLuint colorRenderbuffer;

glGenRenderbuffers(1, &colorRenderbuffer);

glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);

[myContext renderbufferStorage:GL_RENDERBUFFER fromDrawable:myEAGLLayer];

glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
GL_RENDERBUFFER, colorRenderbuffer);
```

Table 3.1.8 Create color Renderbuffer

8. Retrieve the height and width of the color renderbuffer.



```
GLint width;

GLint height;

glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_WIDTH,
&width);

glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_HEIGHT,
&height);
```

Table 3.1.9 Retrieve the height and width

In earlier cases, the width and height of the renderbuffers was explicitly provided to allocate storage for the buffer. Here, the code retrieves the width and height from the color renderbuffer after its storage is allocated. The application does this because the actual dimensions of the color renderbuffer are calculated based on the view's bounds and scale factor. Other renderbuffers attached to the framebuffer must have the same dimensions. In addition to using the height and width to allocate the depth buffer, use them to assign the OpenGL ES viewport as well as to help determine the level of detail required in the application's textures and models.

9. Allocate and attach a depth buffer.

10. Test the framebuffer object.

Offscreen renderbuffer	glRenderbufferStorage
Drawable renderbuffer	renderbufferStorage:fromDrawable:
Texture	glFramebufferTexture2D

Table 3.1.10 "Mechanisms for allocating the color attachment of the framebuffer" [4]

3.2 Drawing to a Framebuffer Object

Once the framebuffer object is created, it should be filled. This section describes the steps required to render new frames and present them to the user. “Rendering to a texture or offscreen framebuffer acts similarly, differing only in how the application uses the final frame” [4].

Generally, applications render new frames in one of two situations:

- On demand; it renders a new frame when it recognizes that the data used to render the frame changed.
- In an animation loop; it assumes that data used to render the frame changes for every frame.

3.2.1 Rendering on Demand

Rendering on demand is appropriate when the data used to render a frame does not change very often, or only changes in response to user action. OpenGL ES on iOS is well suited to this model. When we present a frame, Core Animation caches the frame and uses it until a new frame is presented. By only rendering new frames when you need to, you conserve battery power on the device, and leave more time for the device to perform other actions.

3.2.2 Rendering a Frame

Figure 3.2.1 shows the steps an OpenGL ES application should take on iOS to render and present a frame. These steps include many hints to improve performance in the application.

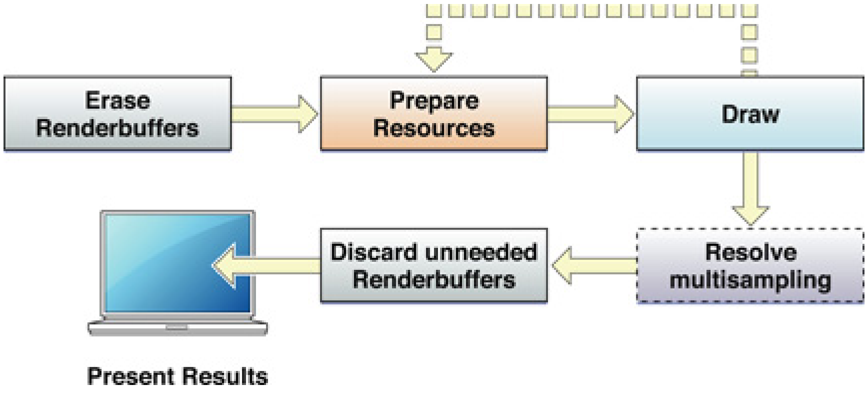
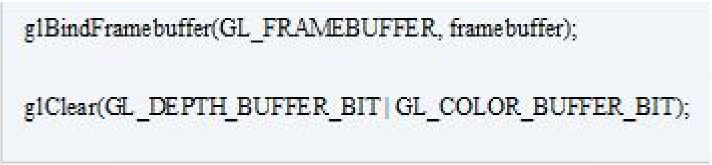


Figure 3.2.1 iOS OpenGL Rendering Steps

1. Erase the Renderbuffers: Erase all the render buffers contents whose contents from a previous frames are not needed to draw the next frame, when starting every frame Call the glClear function, passing in a bit mask with all of the buffers to clear, as shown in table 3.2.1.



```
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);

glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
```

Table 3.2.1 Erasing the renderbuffers

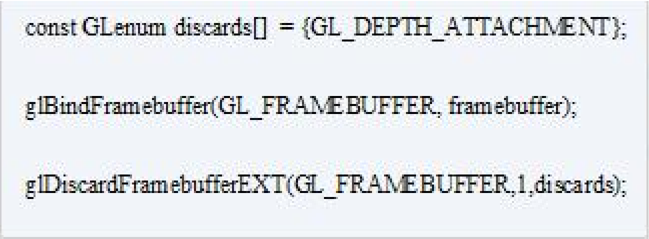
“Not only is using glClear more efficient than erasing the buffers manually, but using glClear hints to OpenGL ES that the existing contents can be discarded. On some graphics hardware, this avoids costly memory operations to load the previous contents into memory “ [4].

2. Prepare OpenGL ES Objects: This step and the next step is the important part of the application, where the application decides what it wants to display to the user. During this step, all the OpenGL ES objects, vertex buffer, textures and other required objects that are needed to render the frame are prepared.

3. Execute Drawing Commands: During this step all the objects prepared in the previouse step are taken and the drawing commands are submitted to use them. "Although the application can alternate between modifying objects and submitting drawing commands (as shown by the dotted line in Figure 3.2.1), it runs faster if it only performs each step once" [4].

4. Discard Unneeded Renderbuffers: EXT_discard_framebuffer extension defines the discard buffer. A discard is a performance hint to OpenGL ES; it tells OpenGL ES that the contents of one or more renderbuffers are not used by the application after the discard command completes. By hinting to OpenGL ES that the application does not need the contents of a renderbuffer, the data in the buffers can be discarded or expensive tasks to keep the contents of those buffers updated can be avoided.

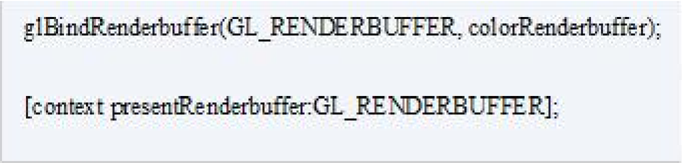
During this stage the application has submitted all of the drawing commands to the frame, in the rendering loop. While the application needs the color renderbuffer to display to the screen, it probably does not need the depth buffer's contents. Table 3.2.2 discards the contents of the depth buffer.



```
const GLenum discards[] = {GL_DEPTH_ATTACHMENT};

glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);

glDiscardFramebufferEXT(GL_FRAMEBUFFER,1,discards);
```

Table 3.2.2 Discarding the depth framebuffer

5. **Present the Results to Core Animation:** At this step, the color renderbuffer holds the completed frame, so all it needs to do is present it to the user Table 3.2.3 binds the renderbuffer to the context and presents it. This causes the completed frame to be handed to Core Animation. It is by default, the application must assume that once the contents of the renderbuffer are presented, it discards the renderbuffer. This means that every time the application presents a frame, it must completely recreate the frame's contents when it renders a new frame. The code above always erases the color buffer for this reason.



```
glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);

[context presentRenderbuffer:GL_RENDERBUFFER];
```

Table 3.2.3 Presenting the finished frame

If the application wants to preserve the contents of the color renderbuffer between frames, add the kEAGLDrawablePropertyRetainedBacking key to the dictionary stored in the drawableProperties property of the CAEAGLLayer object, and remove the GL_COLOR_BUFFER_BIT constant from the earlier glClear function call. The application's performance may be hindered if retained backing is used, as it requires iOS to allocate additional memory to preserve the buffer's contents.

3.3 Subtractive Color Blending

In this application, user can perform a subtractive color blending. Subtractive color blending is similar to the one used in printers, which is also known as CMYK blending. The equation for subtractive color blending is

Cyan + Magenta = Blue

Cyan + Yellow = Green

Magenta + Yellow = Red

Cyan + Magenta + Yellow = Black

This type of blending is obtained by using the glBlendfunc() function. Initially the paint color is specified in RGB, while passing to the glColor4f() function, the colors are mapped to CMYK scheme. In the blend mode the CMYK colors are subtracted with the base color white (1, 1, 1) which in turn forms RGB colors. Setting the colors to the CMYK map and passing the parameters GL_SRC_ALPHA and GL_ONE_MINUS_SRC_COLOR to the glBlendfunc(), a subtractive color blending can be obtained.

3.4 Varying width based on Touch Radius

The radius of the touch is passed to the function that updates the drawing based on the user input. The undocumented version for finding the radius of the touch is used. Touch radius is updated as the user moves the finger from one point to another, this way the touch size varies from the moment the touches begin until touches end. The variable @pathMajorRadius of UITouch is used to identify the radius of the touch.

4. Comparison with current existing products

There are quite a number of applications that are currently available in the market that allows a user to do free hand drawing on touch sensitive devices. Most of the applications lack the ability to take advantage of the certain wonderful functionalities the device has offered. In this project the application developed takes the advantage of those functionalities that enhances the application interface and new features are being implemented.

App Name Functionalities	Doodle	MyPaint	Drawing Box	Draw Cast	SketchBookX	CS298 App
Blurred edge of the brush	NO	NO	YES	NO	YES	YES
Brush size vary with touch	NO	NO	NO	NO	NO	YES
CMYK color Blending	NO	NO	NO	NO	NO	YES
Edit image from the library	YES	NO	YES	YES	YES	YES
Adding colors to the image	NO	NO	NO	NO	NO	YES
Sharing the image online	NO	NO	NO	YES	NO	YES
Color Wheel	NO	NO	NO	NO	YES	YES
Zoom and edit	NO	YES	NO	YES	YES	YES
Tools hidden for full canvas use	NO	YES	YES	NO	YES	YES

Table 4.1 Comparison table

5. Screen shots

The main UI of the Application consists of variety of tools, which includes zoom, pen, eraser, photo library, erase the entire screen, share, and a color wheel.

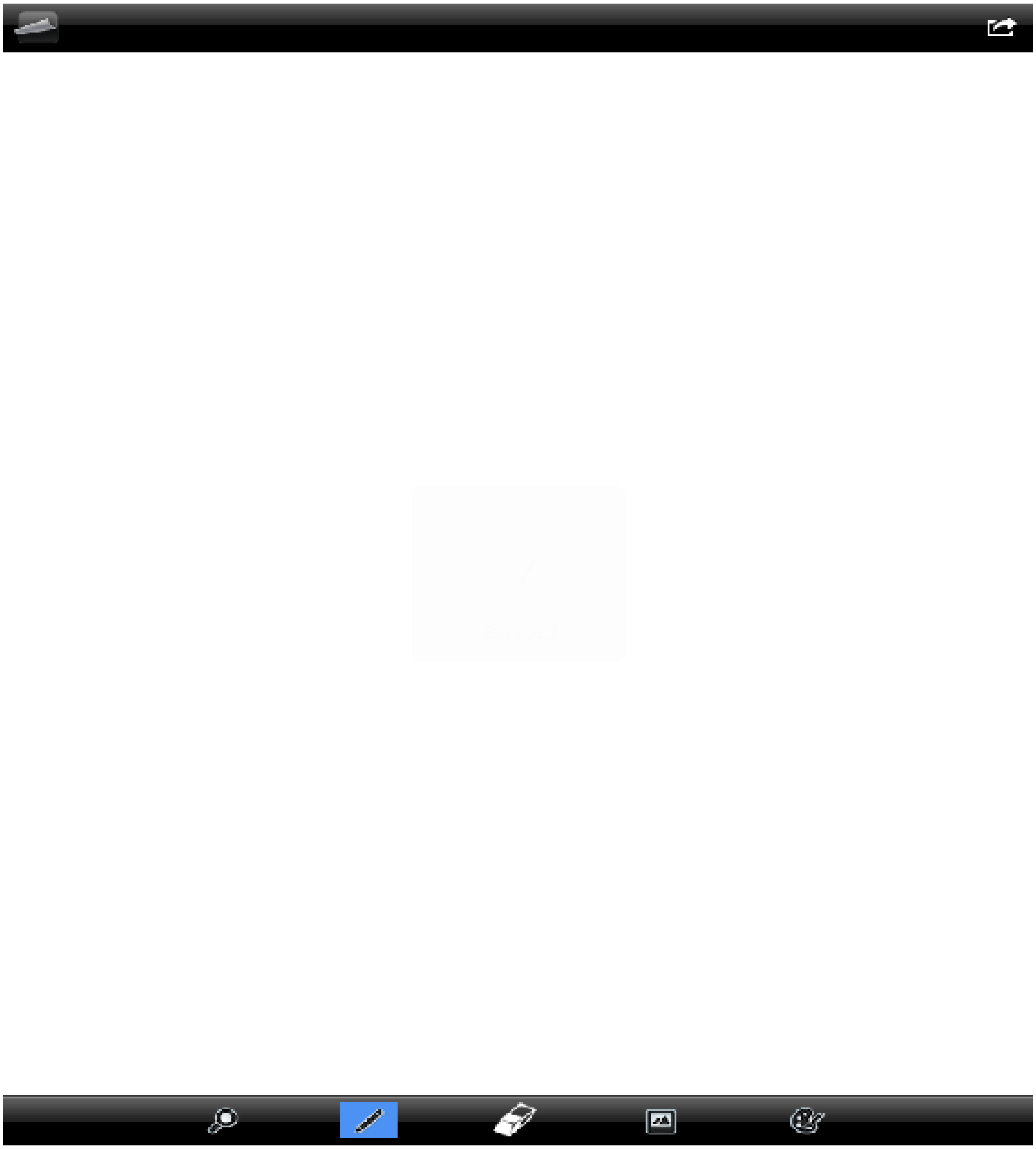
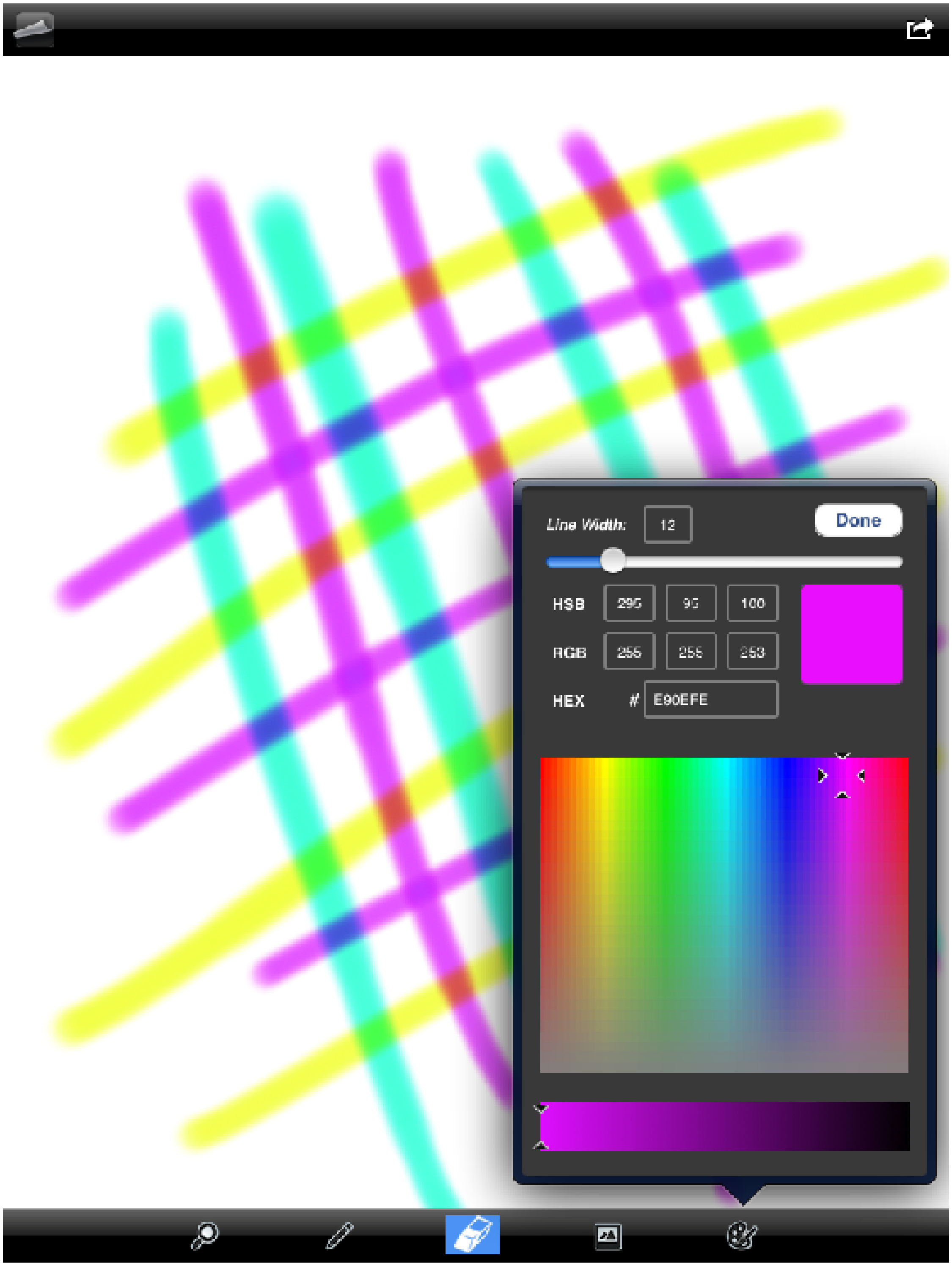


Figure 5.1 Main UI



Line Width: 12

HSB 295 95 100

RGB 255 255 253

HEX # E90EFE

Figure 5.2 Color Wheel and Pen Width Selection

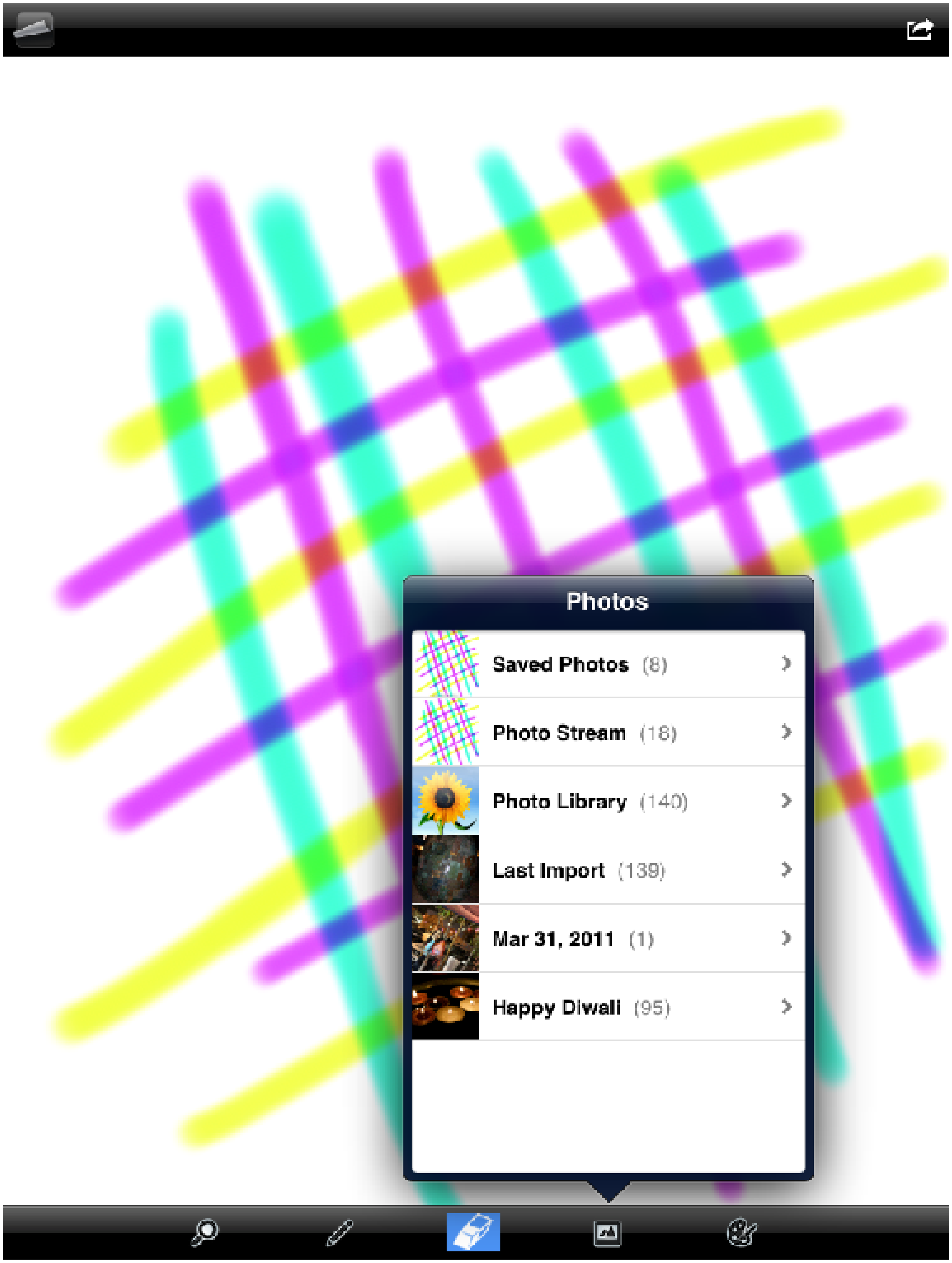
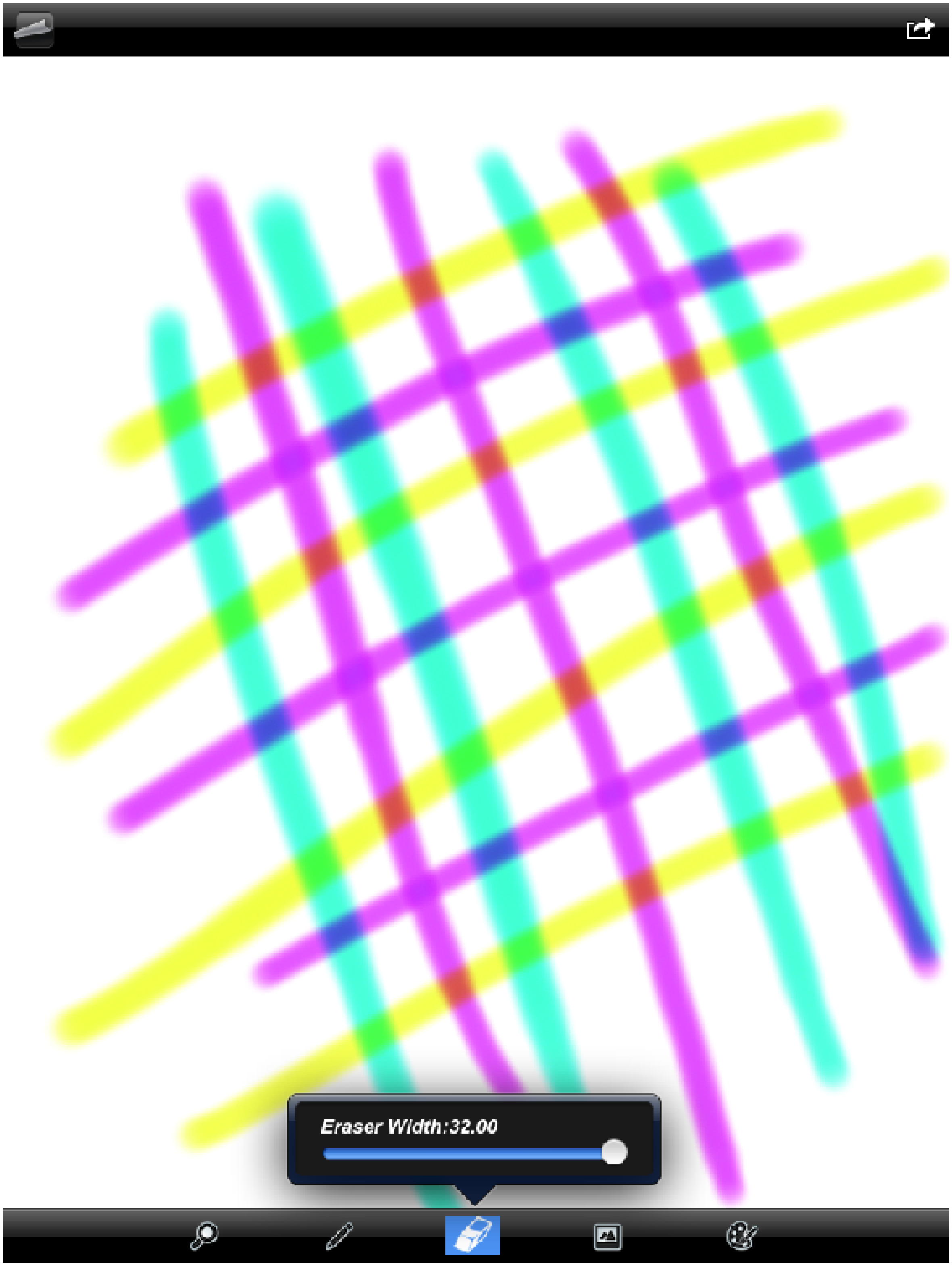


Figure 5.3 Photo Library



Eraser Width:32.00

Figure 5.4 Eraser Width Selection

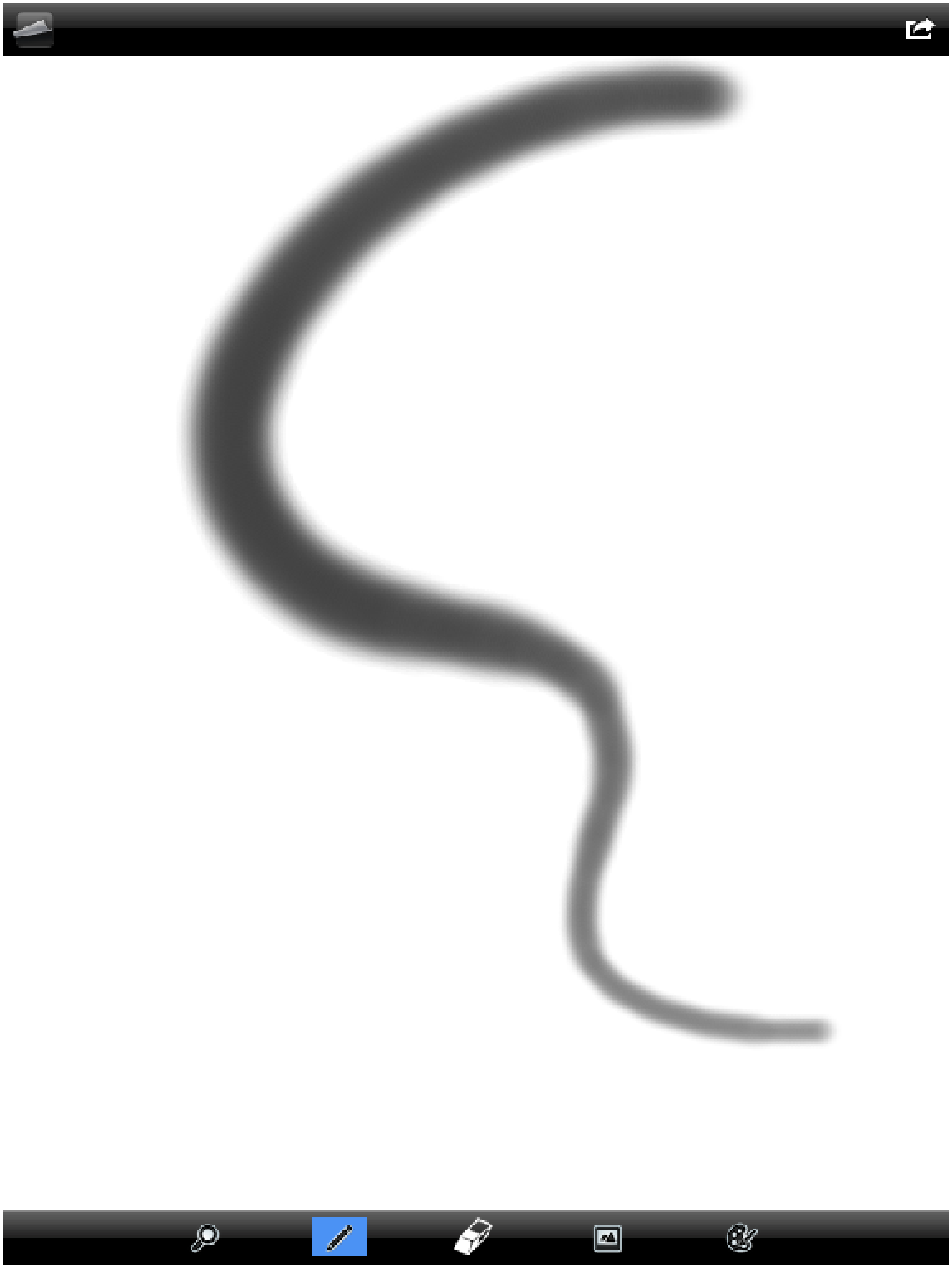


Figure 5.5 Varying width based on the touch radius

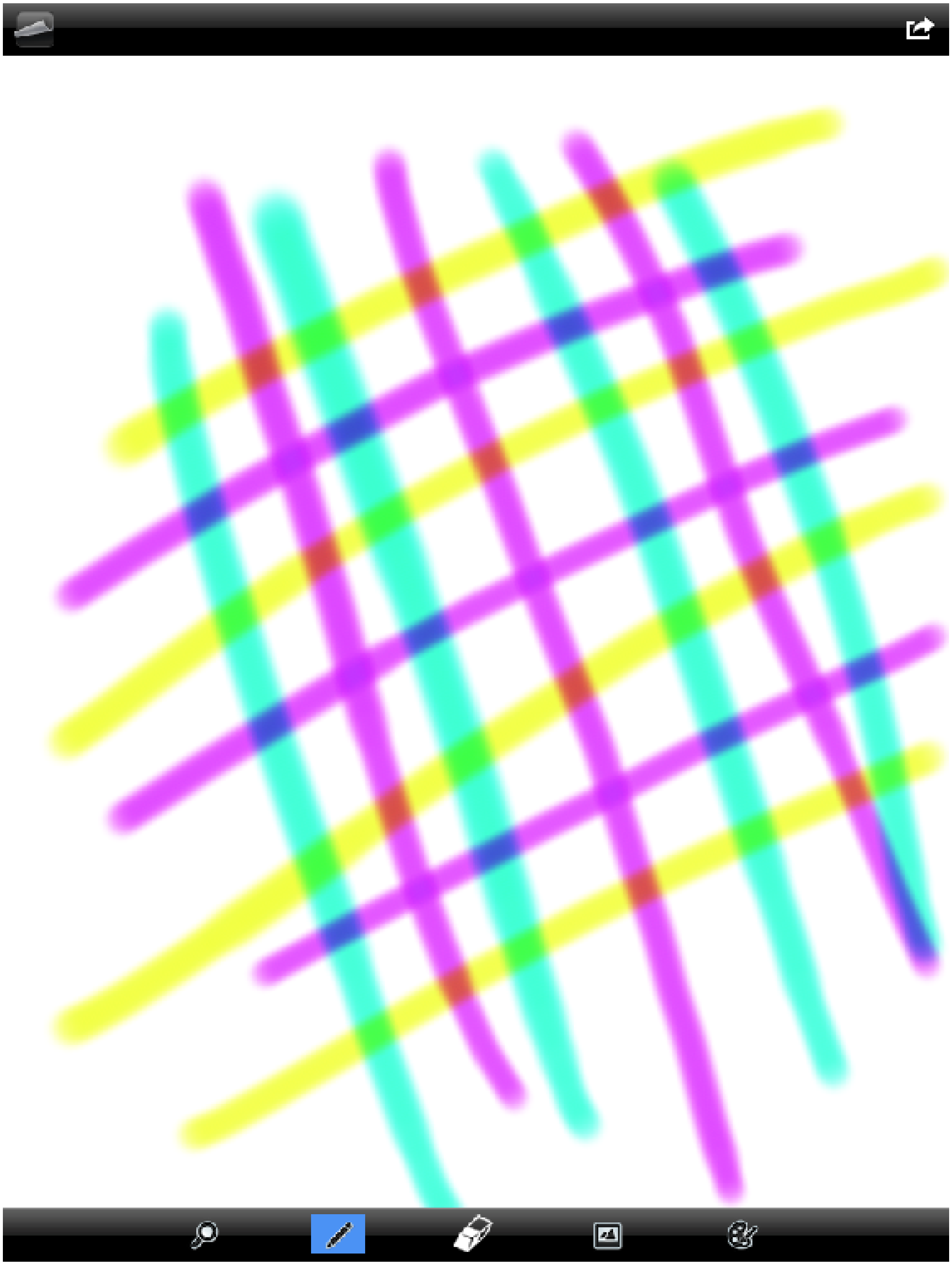
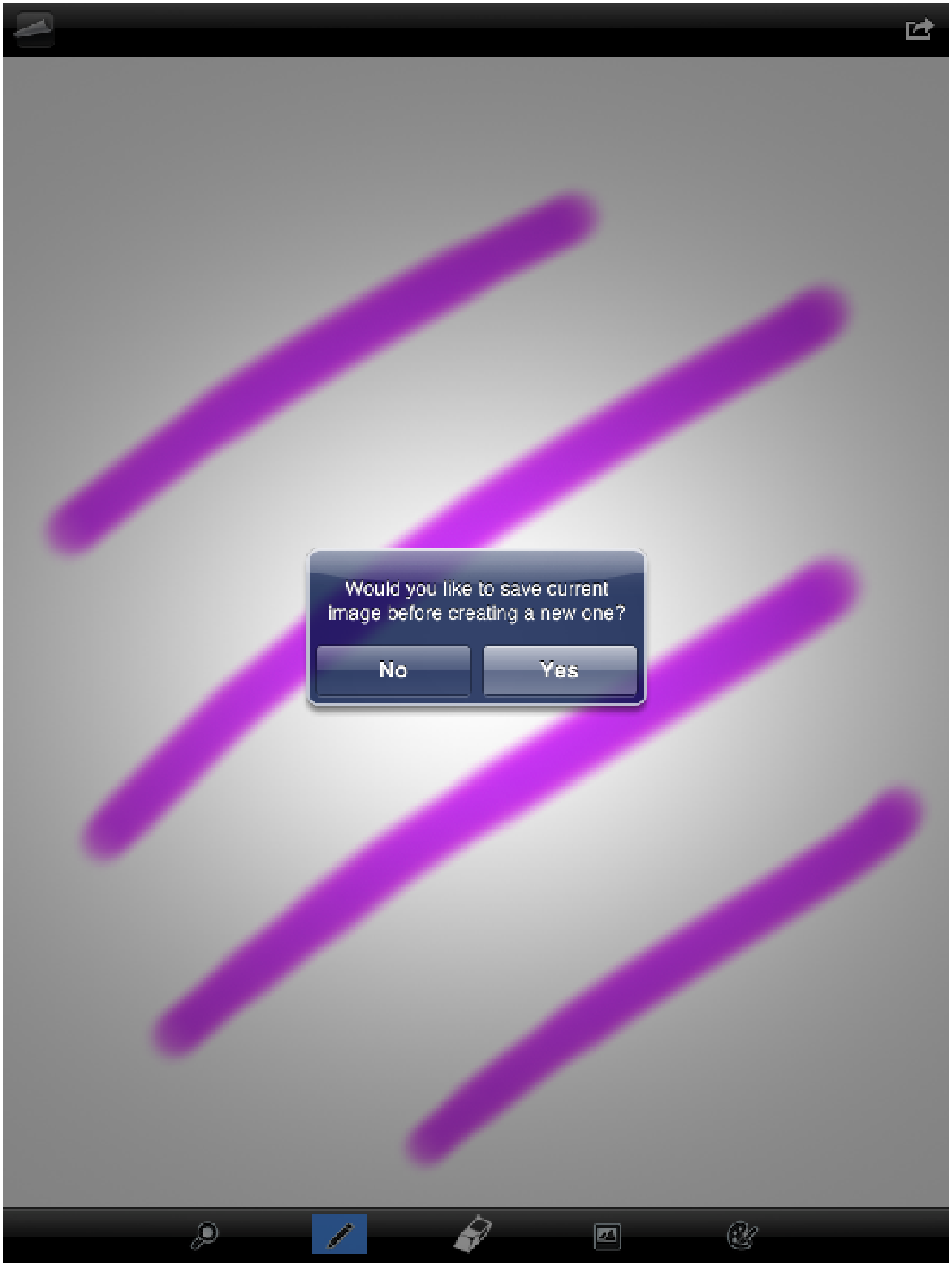


Figure 5.6 Subtractive Color Blending Example.



Would you like to save current image before creating a new one?

Figure 5.7 Notification to save before erasing the drawing

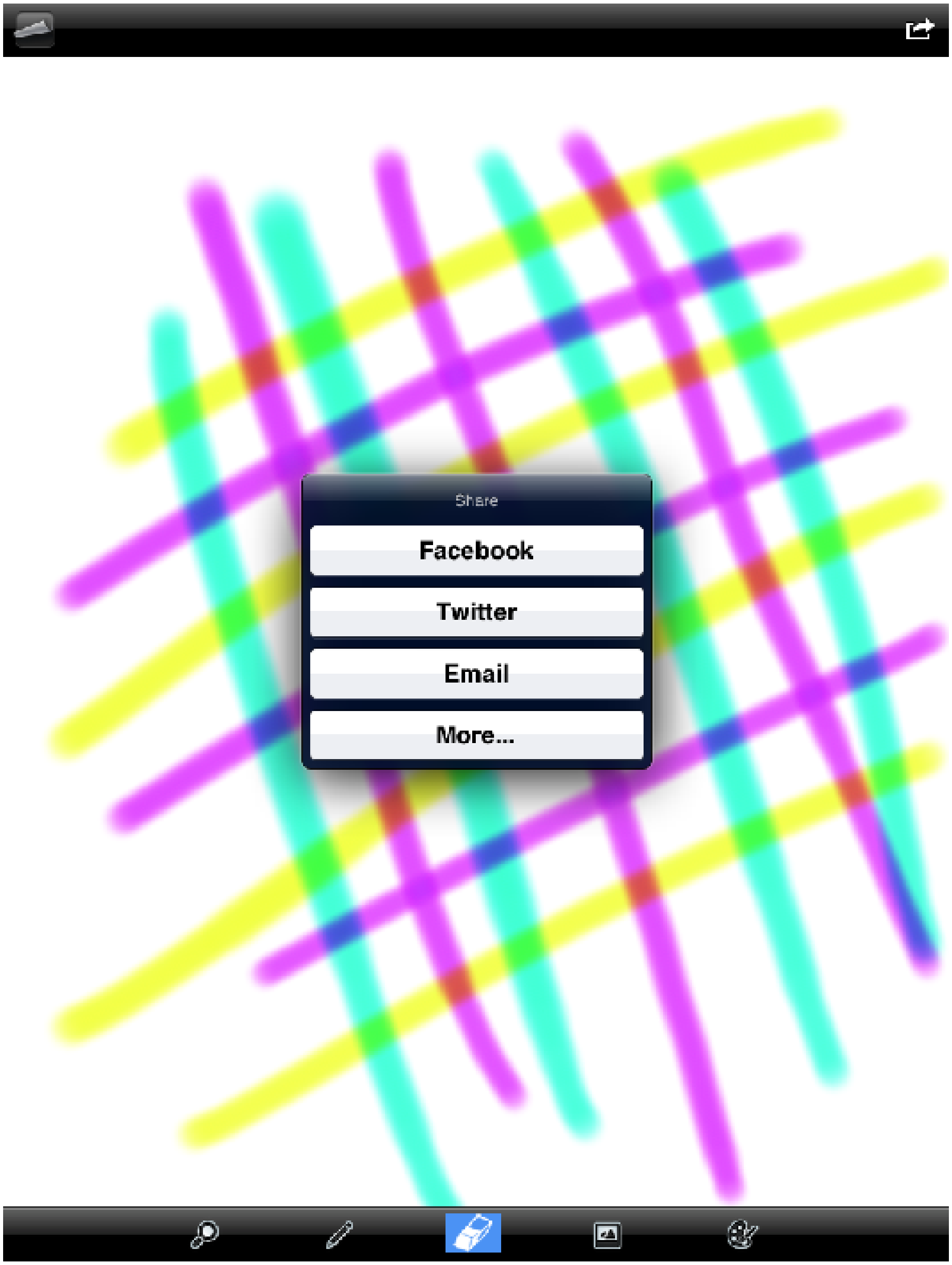


Figure 5.8 sharing the drawing

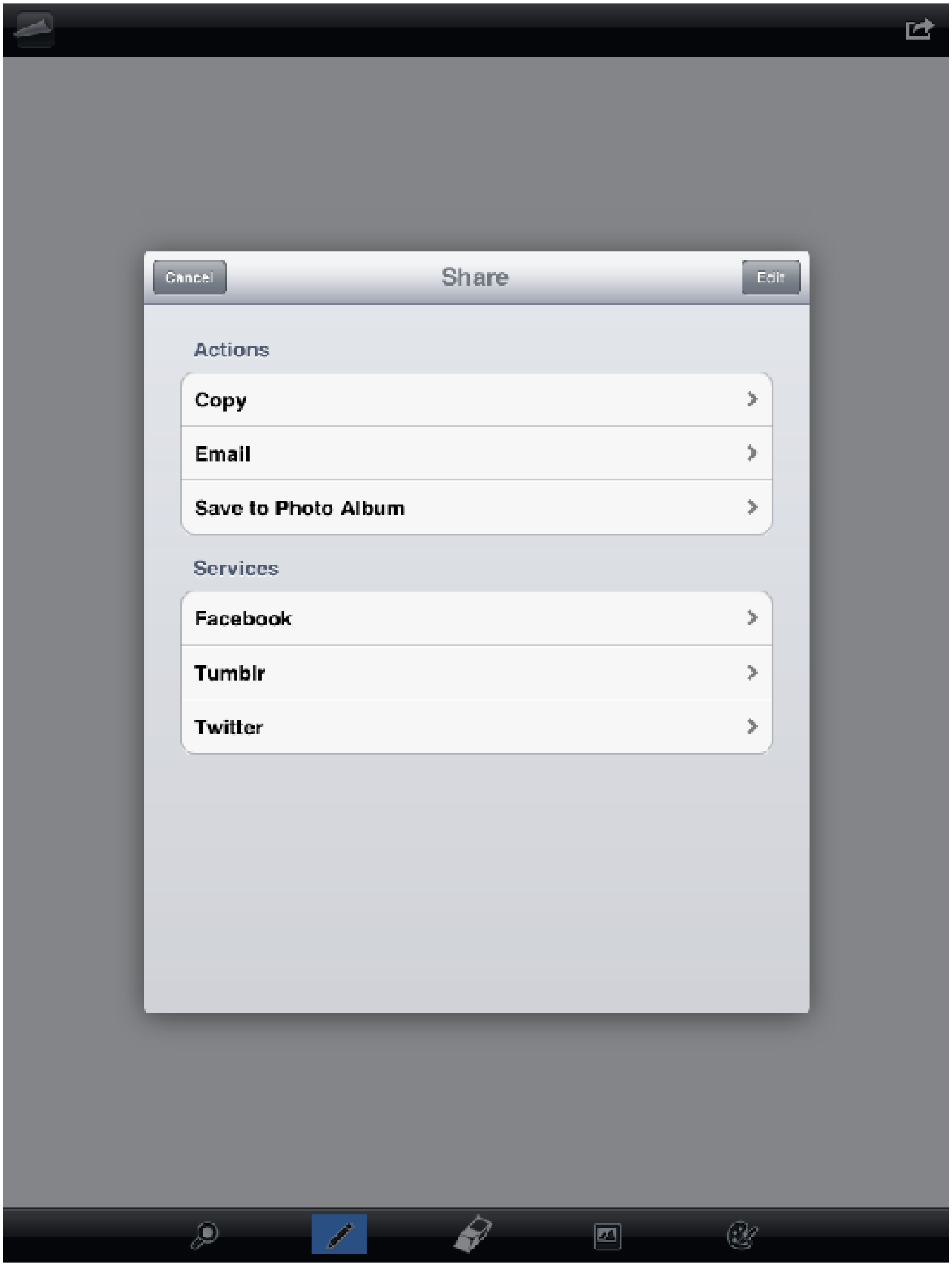
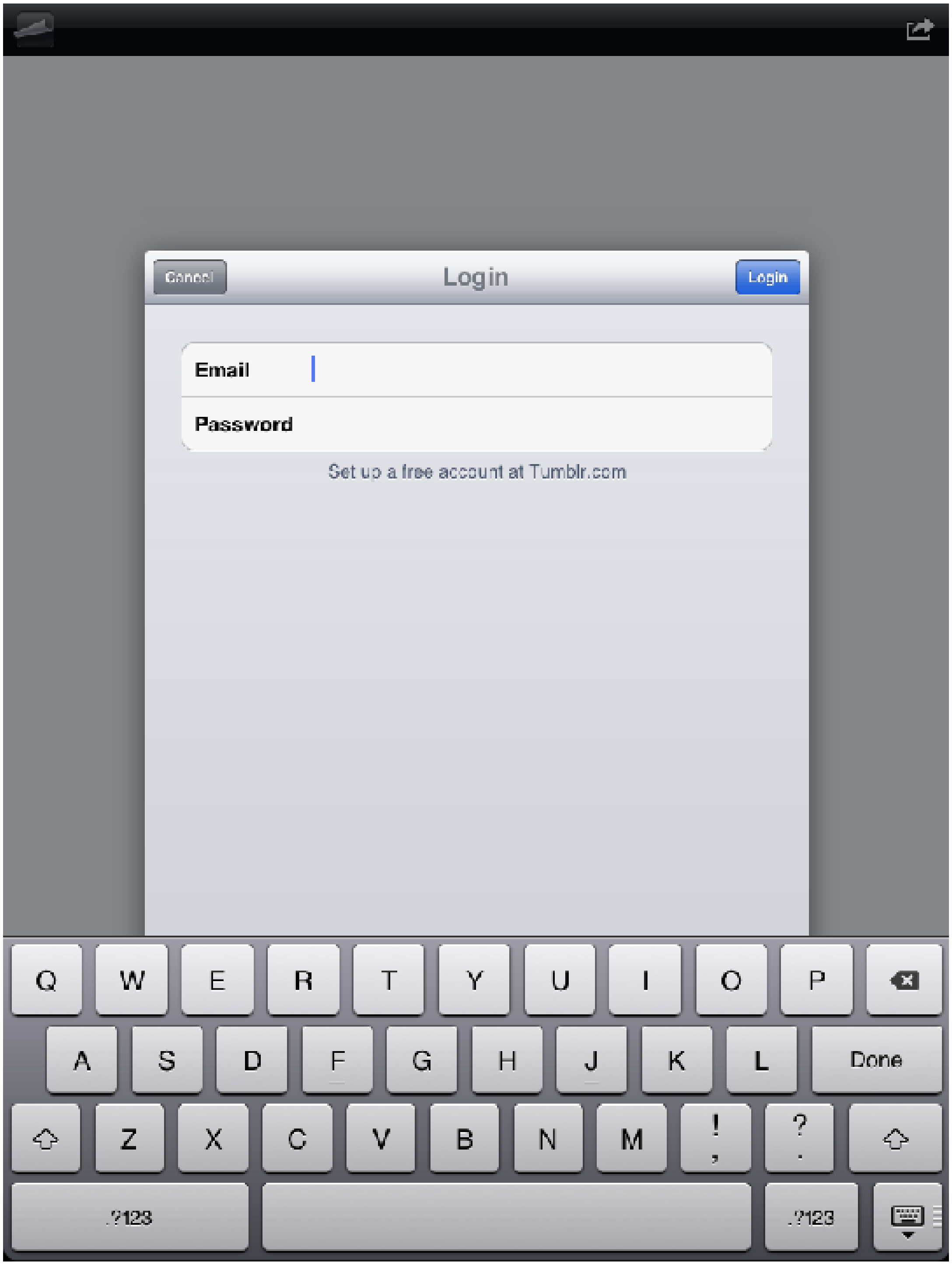


Figure 5.9 Different actions and services that can be performed on the drawing



Set up a free account at Tumblr.com

Figure 5.10 Login window for the Services

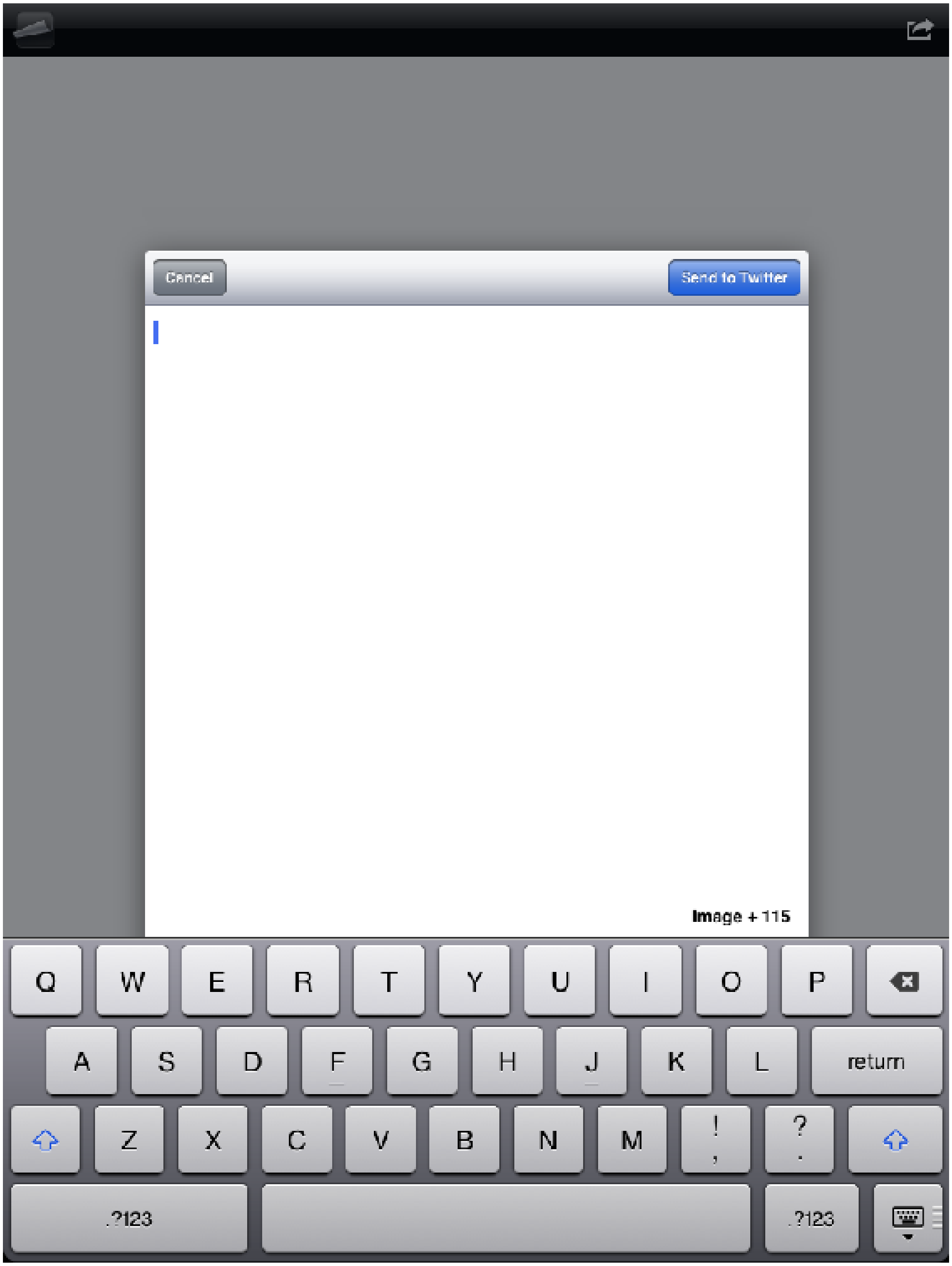


Figure 5.11 Sharing the drawing with a message to a service

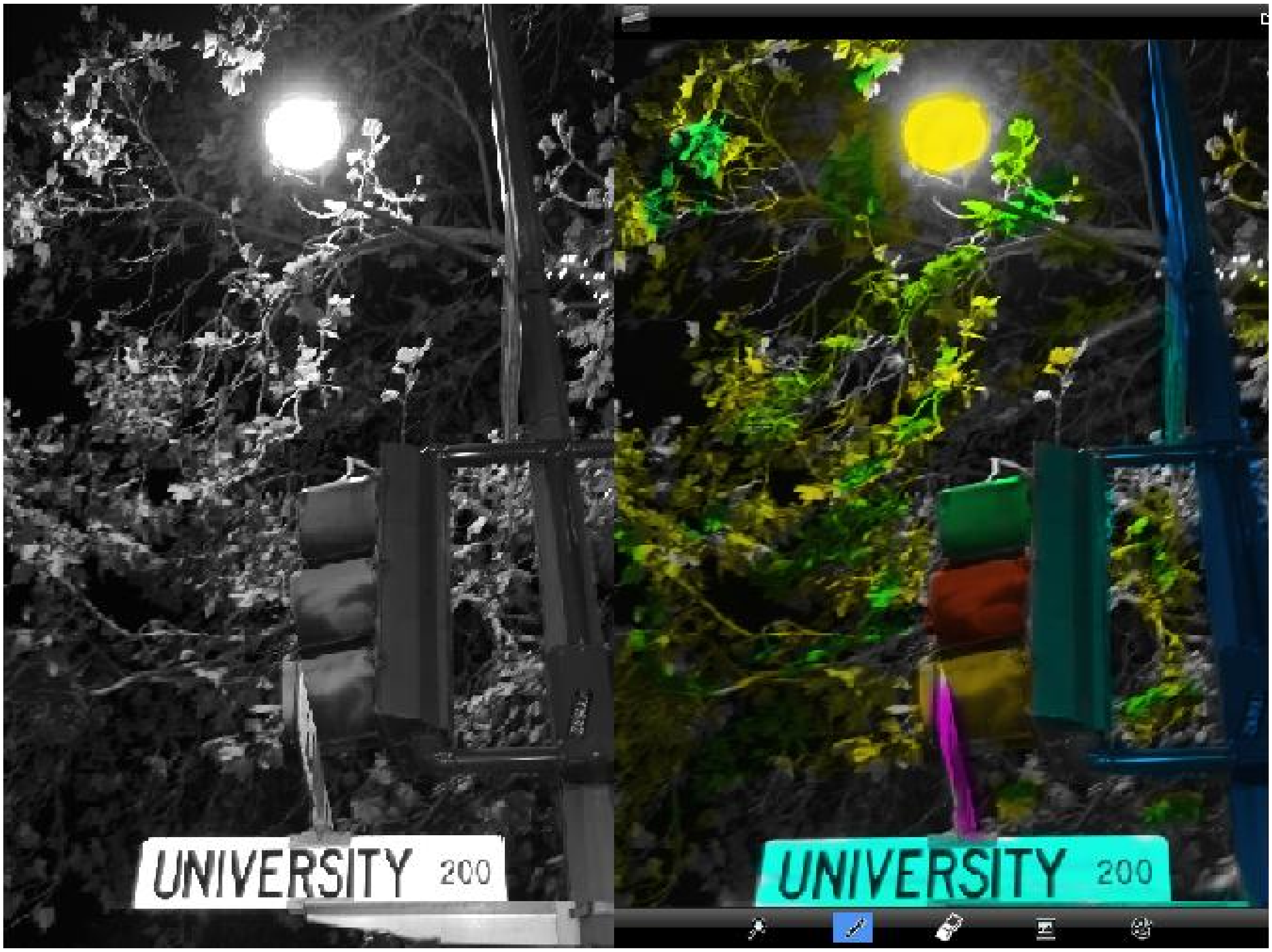


Figure 5.12 Adding color to the image from photo library

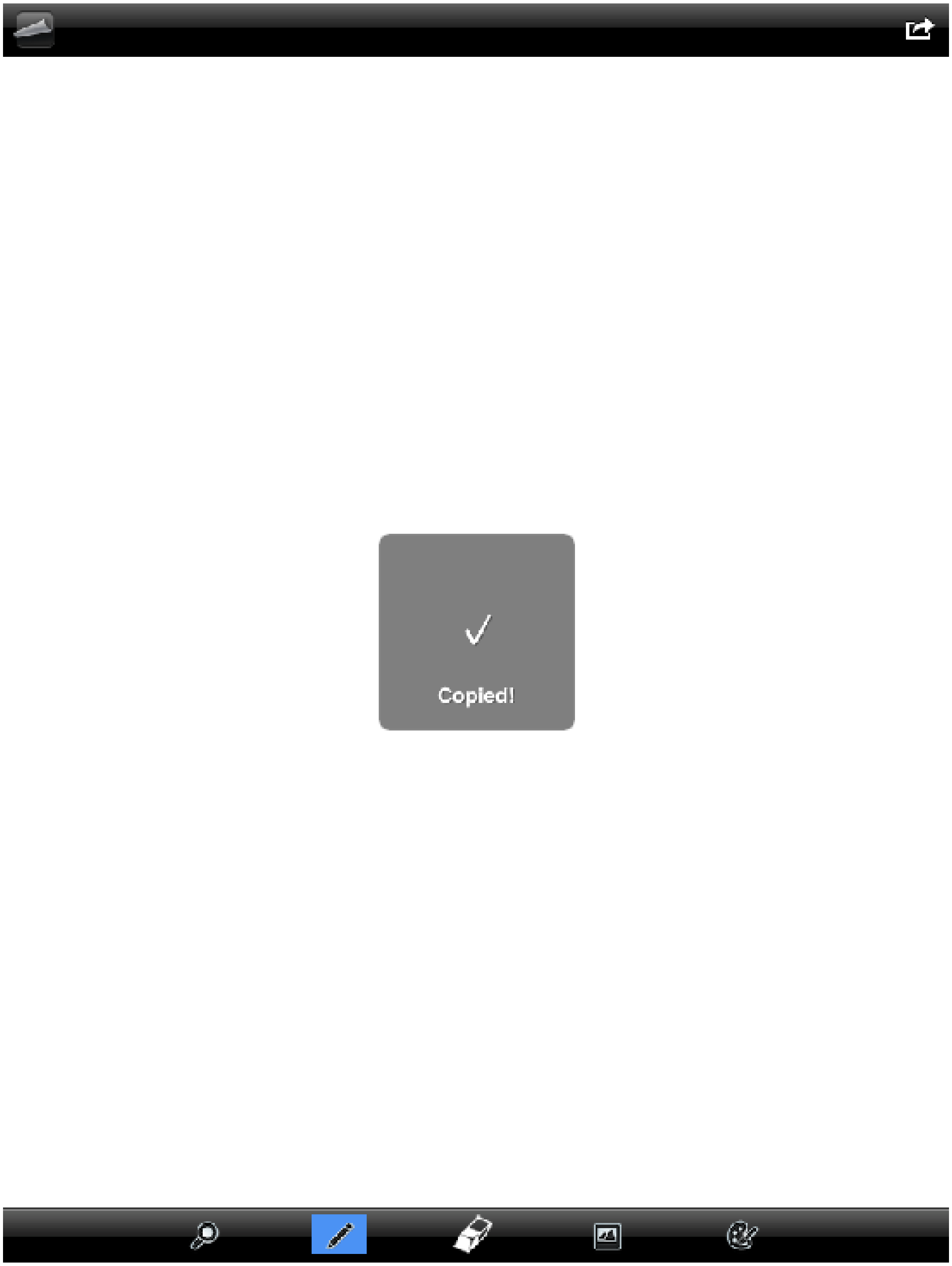


Figure 5.13 Copying image to the device buffer notification

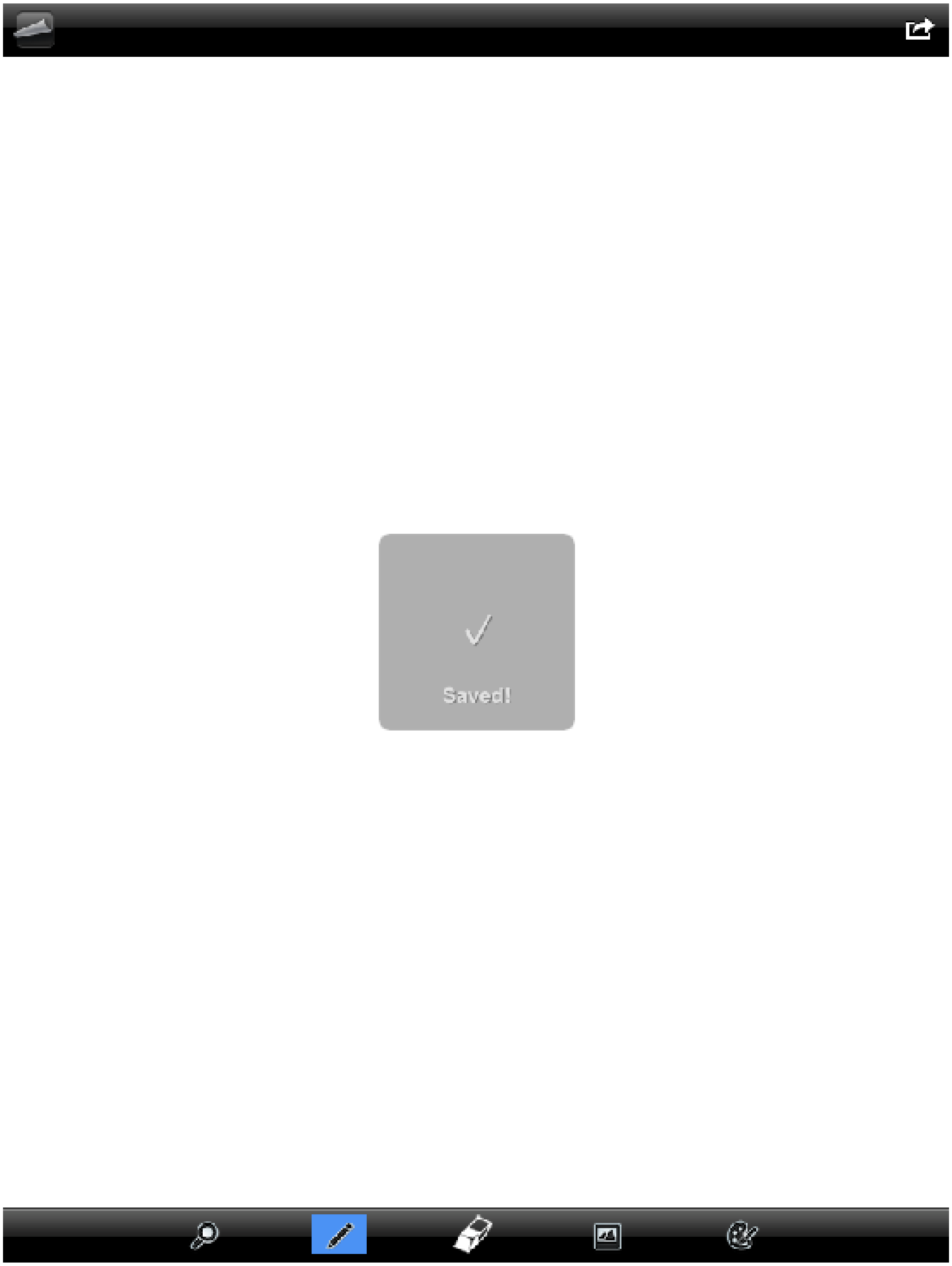


Figure 5.14 Saving the drawing to the device photo library notification

6. Conclusion

Due to the rapid advancement of technology, there are many touch devices in the market available at very low cost. In this project, I have taken advantage of the touch sensitive device to allow the user to draw freely and be able to perform Subtractive color blending. The user can also add an image from the photo library and change the image colors. The capability of the device to identify the touch radius has been taken advantage of, based on which the pen size varies in the single continuous touch.

References

[1] Foley, van Dam, Feiner, and Hughes (2000). Computer Graphics: Principles and Practice in C

[2] Multi-Touch All-Point Touchscreens: The Future of User Interface Design Chitiz Mathema, senior product marketing engineer, Cypress Semiconductor Corp. (http://www.articlesbase.com/technology-articles/multitouch-allpoint-touchscreens-the-future-of-user-interface-de

[3] Apple developer iOS Reference Library https://developer.apple.com/library/ios/navigation/

[4] Guralnick, D. (2006). "How to Design Effective, Motivating User Interfaces." American Society for Training & Development TechKnowledge Conference, Denver, CO.

[5] M. Resnick, B. Myers, K. Nakakoji, B. Shneiderman, R. Pausch, T. Selker, M. Eisenberg, "Design Principles for Tools to Support Creative Thinking", http://www.cs.umd.edu/hcil/CST/Papers/designprinciples.pdf, October 30, 2005.